1934.64567

#2 Priority Paper PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re U.S. Patent Application)
)
Applicant: Tsujimori et al.)
)
Serial No.)
)
Filed: July 26, 2000)
)
For: COMPILER DEVICE AND COMPUTER READABLE RECORDING MEDIUM RECORDED WITH COMPILER PROGRAM)
)
Art Unit:)

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231, on July 26, 2000
Express Label No.: EL409506943US
Signature: L. Daniels
F-CLASS.WCM
Appr. February 20, 1998

Assistant Commissioner for Patents
Washington, DC 20231

Sir:

CLAIM FOR PRIORITY

Applicants claim foreign priority benefits under 35 U.S.C. § 119 on the basis of the foreign application identified below:

Japanese Patent Application No. 11-319570

Filing Date: November 10, 1999

A certified copy of the priority document is enclosed.

Respectfully submitted,

GREER, BURNS & CRAIN, LTD.

By
Patrick G. Burns
Reg. No. 29,367

July 26, 2000
Sears Tower - Suite 8660
233 South Wacker Drive
Chicago, IL 60606
(312) 993-0080

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: １９９９年１１月１０日

出 願 番 号
Application Number: 平成１１年特許願第３１９５７０号

出 願 人
Applicant (s): 富士通株式会社

CERTIFIED COPY OF
PRIORITY DOCUMENT

２０００年 ４月 ７日

特 許 庁 長 官
Commissioner,
Patent Office 近 藤 隆 彦

出証番号 出証特２０００－３０２４５５５

【書類名】　特許願

【整理番号】　9951141

【提出日】　平成11年11月10日

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/44

G06F　9/46

【発明の名称】　コンパイラ装置及びコンパイラプログラムを記録した記録媒体

【請求項の数】　8

【発明者】

【住所又は居所】　神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

【氏名】　辻森　誘二

【発明者】

【住所又は居所】　神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

【氏名】　大脇　斉

【特許出願人】

【識別番号】　000005223

【氏名又は名称】　富士通株式会社

【代理人】

【識別番号】　100078330

【弁理士】

【氏名又は名称】　笹島　富二雄

【電話番号】　03-3508-9577

【手数料の表示】

【予納台帳番号】　009232

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9719433

【プルーフの要否】　要

【書類名】　　明細書

【発明の名称】　コンパイラ装置及びコンパイラプログラムを記録した記録媒体

【特許請求の範囲】

【請求項１】

複数のスレッドを並列処理させるときに、スレッド単位毎に動的に確保されたインターフェース領域を使用して、プログラムにおける手続呼出しを行なうコードを生成することを特徴とするコンパイラ装置。

【請求項２】

スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するコードを生成するコード生成手段と、

ソースプログラム中のインターフェース領域に対する引用を、該コード生成手段により生成されたコードを実行することで確定した先頭番地を使用して引用するコードに変換するコード変換手段と、

を含んで構成されたことを特徴とするコンパイラ装置。

【請求項３】

前記コード生成手段は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するライブラリの呼出しコードを生成する構成である請求項２記載のコンパイラ装置。

【請求項４】

前記コード生成手段は、ユーザの指定に基づいて、インターフェース領域の先頭番地を確定するコードを生成する構成である請求項２又は請求項３に記載のコンパイラ装置。

【請求項５】

複数のスレッドを並列処理させるときに、スレッド単位毎に動的に確保されたインターフェース領域を使用して、プログラムにおける手続呼出しを行なうコードを生成する機能を実現するためのコンパイラプログラムを記録した記録媒体。

【請求項６】

スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するコードを生成するコード生成機能と、

ソースプログラム中のインターフェース領域に対する引用を、該コード生成機能により生成されたコードを実行することで確定した先頭番地を使用して引用するコードに変換するコード変換機能と、

を実現するためのコンパイラプログラムを記録した記録媒体。

【請求項７】

前記コード生成機能は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するライブラリの呼出しコードを生成する構成である請求項６記載のコンパイラプログラムを記録した記録媒体。

【請求項８】

前記コード生成機能は、ユーザの指定に基づいて、インターフェース領域の先頭番地を確定するコードを生成する構成である請求項６又は請求項７に記載のコンパイラプログラムを記録した記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、ソースプログラムをコンパイルするコンパイラ技術に関し、特に、並列処理においてインターフェース領域を使用した手続呼出しを実現する技術に関する。

【０００２】

【従来の技術】

従来から、プログラムにおける手続間のインターフェースをとる方法として、引数を渡す方法、インターフェース領域を使用する方法、外部ファイルを使用する方法がある。なお、「インタフェース領域」とは、例えば、FORTRANにおける「COMMON文」やC, C++における「extern文」を使用して割り当てられた共通ブロック領域や外部変数領域のことをいう。

【０００３】

ここで、具体例として、FORTRANにおける「COMMON文」を使用して、メインルーチンとサブルーチンとの間のインターフェースをとる場合を考察する。メインルーチンでCOMMON宣言がされている共通ブロック名が、サブルーチンでも同様に

宣言されていると、メインルーチンで設定された値がサブルーチンで参照できたり、サブルーチンで設定された値がメインルーチンで参照することができるようになる。

【０００４】

【発明が解決しようとする課題】

しかし、従来の「COMMON文」は、逐次処理で実行されることを前提としていたため、実行可能プログラム単位毎に１つの共通領域が静的に割り当てられていた。このため、このまま「COMMON文」を並列処理に適用すると、次のような不具合が発生してしまうおそれがあった。

【０００５】

即ち、メインルーチンから２つのサブルーチンを呼出し、夫々のサブルーチンを異なったCPUで同時に並列処理し、サブルーチンで値を更新している場合には、各サブルーチンの実行タイミングの差によって、インタフェース領域に保持されている値が不定になることがある。この場合、入力データが同一であっても、演算結果が同一になるとは限らず、プログラムの信頼性が低く、並列処理では「COMMON文」が使用できなかった。

【０００６】

また、C,C++における「extern文」であっても、同様な問題が生じていた。

そこで、本発明は以上のような従来の問題点に鑑み、スレッド単位毎に１つの共通領域を動的に確保することにより、並列処理においてインターフェース領域を使用した手続呼出しを実現したコンパイラ技術を提供することを目的とする。

【０００７】

【課題を解決するための手段】

このため、請求項１記載の発明は、複数のスレッドを並列処理させるときに、スレッド単位毎に動的に確保されたインターフェース領域を使用して、プログラムにおける手続呼出しを行なうコードを生成するコンパイラ装置であることを特徴とする。

【０００８】

ここで、「インターフェース領域」とは、例えば、FORTRANにおける「COMMON

文」やC,C++における「extern文」を使用して割り当てられた共通ブロック領域や外部変数領域のことをいう。これらの共通ブロック領域や外部変数領域を使用して、プログラムにおける手続間のインターフェースがとられる。

【０００９】

かかる構成によれば、プログラム実行時に、スレッド単位毎に動的に確保されたインターフェース領域を使用して、プログラムにおける手続呼出しが行なわれる。このため、複数のスレッドが並列処理されるときであっても、各スレッドにおけるインターフェース領域が独立しているため干渉することがなく、従来技術では不可能であった並列処理におけるインターフェース領域を使用した手続呼出しが実現される。

【００１０】

請求項２記載の発明は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するコードを生成するコード生成手段と、ソースプログラム中のインターフェース領域に対する引用を、該コード生成手段により生成されたコードを実行することで確定した先頭番地を使用して引用するコードに変換するコード変換手段と、を含んでコンパイラ装置を構成したことを特徴とする。

【００１１】

かかる構成によれば、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するコードが生成され、ソースプログラム中のインターフェース領域に対する引用が、生成されたコードを実行することで確定した先頭番地を使用して引用するコードに変換される。従って、複数のスレッドが並列処理されるときであっても、各スレッドにおけるインターフェース領域が独立しているため干渉することがなく、従来技術では不可能であった並列処理におけるインターフェース領域を使用した手続呼出しが実現される。

【００１２】

請求項３記載の発明は、前記コード生成手段は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するライブラリの呼出しコードを生成する構成であることを特徴とする。

【００１３】

ここで、「ライブラリ」とは、標準化されたルーチン及びサブルーチンの集まりであって、プログラム実行時に動的に呼出される動的ライブラリ（Dynamic Linked Library）と、リンカによってオブジェクトプログラムと結合される静的ライブラリ（Static Library）とがある。

【００１４】

かかる構成によれば、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地は、ライブラリを呼出すコードを実行することで確定される。従って、ソースプログラム中にインターフェース領域の先頭番地を確定するための処理を記述する必要はなく、プログラム構造が簡単になる。

【００１５】

請求項４記載の発明は、前記コード生成手段は、ユーザの指定に基づいて、インターフェース領域の先頭番地を確定するコードを生成する構成であることを特徴とする。

【００１６】

かかる構成によれば、ユーザが特定の処理について並列処理を行ないたくない場合には、その旨を指定することで、その処理部分に関しては並列化のためのコードが生成されなくなる。従って、ユーザの考えに即した、柔軟なプログラム構造が提供される。

【００１７】

請求項５記載の発明は、複数のスレッドを並列処理させるときに、スレッド単位毎に動的に確保されたインターフェース領域を使用して、プログラムにおける手続呼出しを行なうコードを生成する機能を実現するためのコンパイラプログラムを記録媒体に記録したことを特徴とする。

【００１８】

ここで、「記録媒体」とは、各種情報を確実に記録でき、かつ、必要に応じて確実に取り出し可能なものをいい、磁気テープ，磁気ディスク，磁気ドラム，ＩＣカード，ＣＤ－ＲＯＭ等の可搬記録媒体が該当する。

【００１９】

かかる構成によれば、プログラム実行時に、スレッド単位毎に動的に確保され

たインターフェース領域を使用して、プログラムにおける手続呼出しを行なうコードを生成する機能を実現するコンパイラプログラムが記録媒体に記録される。従って、かかるコンパイラプログラムを記録した記録媒体があれば、一般的なコンピュータシステムを利用して、本発明に係るコンパイラ装置を容易に構築することが可能となる。

【００２０】

請求項６記載の発明は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するコードを生成するコード生成機能と、ソースプログラム中のインターフェース領域に対する引用を、該コード生成機能により生成されたコードを実行することで確定した先頭番地を使用して引用するコードに変換するコード変換機能と、を実現するためのコンパイラプログラムを記録媒体に記録したことを特徴とする。

【００２１】

かかる構成によれば、コード生成機能と、コード変換機能と、を実現するためのコンパイラプログラムが記録媒体に記録される。従って、かかるコンパイラプログラムを記録した記録媒体があれば、一般的なコンピュータシステムを利用して、本発明に係るコンパイラ装置を容易に構築することが可能となる。

【００２２】

請求項７記載の発明は、前記コード生成機能は、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地を確定するライブラリの呼出しコードを生成する構成であることを特徴とする。

【００２３】

かかる構成によれば、スレッド単位毎に動的に確保されたインターフェース領域の先頭番地は、ライブラリを呼出すコードを実行することで確定される。従って、ソースプログラム中にインターフェース領域の先頭番地を確定するための処理を記述する必要はなく、プログラム構造が簡単になる。

【００２４】

請求項８記載の発明は、前記コード生成機能は、ユーザの指定に基づいて、インターフェース領域の先頭番地を確定するコードを生成する構成であることを特

徴とする。

【００２５】

かかる構成によれば、ユーザが特定の処理について並列処理を行ないたくない場合には、その旨を指定することで、その処理部分に関しては並列化のためのコードが生成されなくなる。従って、ユーザの考えに即した、柔軟なプログラム構造が提供される。

【００２６】

【発明の実施の形態】

以下、添付された図面を参照して本発明を詳述する。

先ず、実施の形態を説明する前に、本発明の内容の理解を容易にすることを目的として、従来技術の問題点及び本発明の概要を説明する。

【００２７】

図９は、FORTRAN上でOpenMPを使用することを前提として、複数の手続（サブルーチンXAXIS，YAXIS）を並列処理させるための概要を示す。即ち、図９（Ａ）に示すような逐次処理のソースプログラムを、並列処理に変更すべく、図９（Ｂ）に示すようなソースプログラムに書き換える。ここで、ソースプログラム中の「!$OMP」は、コンパイラオプションの設定により、その行を有効／無効にすることを制御できる文である。また、「PARALLEL SECTIONS」と「END PARALLEL SECTIONS」とで囲まれた範囲に記述された「SECTION」により、並列処理であることが指定される。

【００２８】

そして、かかるソースプログラムを実行すると、図９（Ｃ）に示すように、CPU１及びCPU２において、メインルーチンからサブルーチンXAXIS及びYAXISが夫々コールされる。そして、メインルーチンとサブルーチンXAXIS及びYAXISとの間のインターフェースとして、COMMON文によりプログラム単位毎に静的領域に１つだけ確保された変数ａが使用される場合を考える。この場合、変数ａは、プログラム単位毎に静的領域に１つだけ確保される。従って、サブルーチンXAXIS及びYAXISは、共通した静的領域に保持された変数ａを使用することとなる。このため、サブルーチンXAXISとYAXISとにおいて、変数ａの参照後に変数ａの値の更新があ

る場合、CPUの処理タイミングによっては、一方のサブルーチンで変数ａに設定された数値を処理する前に、他方のサブルーチンでその数値が更新されることがあり、夫々のサブルーチンにおける演算結果がユーザの予想した値と異なってしまうおそれがある。従って、従来技術では、各サブルーチンで処理された演算結果が不定となってしまい、並列処理に対応することができなかったのである。

【００２９】

このため、本願発明では、COMMON文を用いて確保する領域をスレッド単位毎に動的に１つ確保、即ち、スレッド単位毎にプライベート化するようにして、並列処理に対応可能にするようにした。

【００３０】

次に、本発明に係るコンパイラ装置（以下「コンパイラ装置」という）の構成について、図１を参照しつつ説明する。

コンパイラ装置１０は、ソース解析部１２と、最適化部１４と、コード生成部１６と、を含んで構成される。ソース解析部１２は、ソースプログラムを解析して中間言語を生成する機能を提供し、後述する処理により並列処理への対応を実現する。最適化部１４は、ソース解析部１２により生成された中間言語に対して、例えば、処理時間を短縮するような最適化を行う機能を提供する。コード生成部１６は、最適化部１４により最適化された中間言語を読み込み、アセンブラコードまたはオブジェクトコードを出力する機能を提供する。ここで、コード生成部１６によりアセンブラコードが出力された場合には、アセンブラコンパイラによりアセンブラコードをオブジェクトコードにコンパイルする必要がある。そして、コード生成部１６又はアセンブラコンパイラにより出力又はコンパイルされたオブジェクトコードは、図示しないリンカによって結合され、実行可能プログラムが完成する。

【００３１】

なお、コンパイラ装置１０は、少なくともメモリと中央処理装置（ＣＰＵ）とを備えたコンピュータ上に構築され、メモリにロードされたプログラムに従ってコンパイル処理が行なわれる。

【００３２】

図２及び図３は、コンパイラ装置１０のソース解析部１２において行なわれるコンパイル処理のフローチャートを示す。

ステップ１（図では「Ｓ１」と略記する。以下同様）では、COMMON文により確保された共通ブロックをスレッド単位でプライベート化することが、例えば、コンパイルオプションにより指定されているか否かが判定される。そして、スレッド単位でプライベート化することが指定されていればステップ２へと進み（Ｙｅｓ）、指定されていなければ処理を終了する（Ｎｏ）。

【００３３】

ステップ２では、ソースプログラムから、COMMON文により宣言された共通ブロック名が取り出される。ここで、ソースプログラムから共通ブロック名が取り出せなかった場合、具体的には、共通ブロック名の取り出しが完了した場合には、その旨を示す識別子等が取り出される。

【００３４】

ステップ３では、取り出された共通ブロック名に基づいて、共通ブロック名の取り出しが完了したか否かが判定される。そして、共通ブロック名の取り出しが完了したならばステップ７へと進み（Ｙｅｓ）、完了していなければステップ４へと進む（Ｎｏ）。

【００３５】

ステップ４では、取り出された共通ブロック名により特定される共通ブロックに対して、スレッド単位でプライベート化する指定があるか否かが判定される。そして、プライベート化する指定があればステップ５へと進み（Ｙｅｓ）、指定がなければステップ２へと戻る（Ｎｏ）。

【００３６】

ステップ５では、例えば、DATA文の有無に基づいて、共通ブロックに対して初期値の設定があるか否かが判定される。そして、初期値の設定があればステップ６へと進み（Ｙｅｓ）、初期値の設定がなければステップ２へと戻る（Ｎｏ）。

【００３７】

ステップ６では、ユーザが指定した共通ブロック名に対して、実行時に初期値があったか否かを判定するための識別子（例えば、＿＿）が付加され、識別子が

付加された共通ブロック名で共通ブロックが生成される。また、共通ブロックの変数等（以下「共通ブロック要素」という）に対して、初期値があったことを示す識別子（例えば、'###-Include initialized values-###'）が設定される。その後、ステップ２へと戻る。

【００３８】

ここで、ソースプログラムが、

```
common / commom#block#name#1 / var1
common / common#block#name#2 / var2
data  var2 /10/
```

である場合には、ステップ６の処理により、

```
common / common#block#name#1 / var1
common / common#block#name#2 / var2
data  var2 /10/
common / ##common#block#name#1 / var1#generate
common / ##common#block#name#2 / var2#generate
character*35  var1#generate, var2#generate
data  var2#generate /'###-Include initialized values-###'/
```

のようなコードが生成される。

【００３９】

以上説明したステップ１～ステップ６の処理によれば、プライベート化指定があって初期値の設定がある共通ブロック名に対して、初期値があることを示す識別子が付加される。また、その共通ブロック要素に対して、初期値が設定されていることを示す識別子が設定される。このようにすれば、後述する実行時の処理によって、動的に確保された共通ブロックに対して動的な初期値の設定が可能となる。

【００４０】

ステップ７では、ソースプログラム中に、スレッド単位でプライベート化する共通ブロックが１つでも存在しているか否かが判定される。そして、プライベート化する共通ブロックが存在していればステップ８へと進み（Ｙｅｓ）、存在し

ていなければ処理を終了する（Ｎｏ）。

【００４１】

ステップ８では、ソースプログラムの入口に、そのソースプログラムで引用される共通ブロックのベース番地（先頭番地）を確定するライブラリ呼出しが生成される。即ち、共通ブロックは、スレッド単位毎に動的に確保されるため、ライブラリから返却されたベース番地を基に変数等をアクセスするように、コードが変換されるのである。また、ライブラリ呼出しは、例えば、次のようなものである。

【００４２】

```
Call Fortran#lib('common#block#name#1',
                 length('common#block#name#1'),
                 var1#generate,
                 &var1,
                 size(var1),
                 &base#of#common#block#name#1)
```

ここで、「common#block#name#1」は共通ブロック名、「length('common#block#name#1')」は共通ブロック名の長さ、「var1#generate」は初期値フラグ、「&var1」は共通ブロックの先頭番地、「size(var1)」は共通ブロックの大きさ、「&base#of#common#block#name#1」はライブラリから返却される共通ブロックをアクセスするためのベース番地である。

【００４３】

なお、ステップ８の処理が、インターフェース領域の先頭番地を確定するコード生成手段及びコード生成機能に該当する。

ステップ９では、ソースプログラムから、実行文中の共通ブロック要素の引用が取り出される。

【００４４】

ステップ１０では、取り出された共通ブロック要素がソースプログラムの最後の引用であるか否かが判定される。そして、その共通ブロック要素が最後の引用であれば処理を終了し（Ｙｅｓ）、最後のものでなければステップ１１へと進む

（Ｎｏ）。

【００４５】

ステップ１１では、取り出された共通ブロック要素が、スレッド単位でプライベート化された共通ブロック要素であるか否かが判定される。そして、プライベート化された共通ブロック要素であればステップ１２へと進み（Ｙｅｓ）、プライベート化された共通ブロック要素でなければステップ９へと戻る（Ｎｏ）。

【００４６】

ステップ１２では、共通ブロック要素の引用が、ライブラリによって確定されたベース番地を使用して引用する形式に変換される。ベース番地を使用した共通ブロック要素の引用は、例えば、ユーザプログラムが、

```
COMMON  /COM/COM#VAR
COM#VAR = 1
```

となっていた場合、これをライブラリ呼出しを使用して

```
call Fortran#lib(……, &base)
base->VAR = 1
```

というように変換される。ここで、「->」はポインタ演算を示す。

【００４７】

なお、ステップ１２の処理が、コード変換手段及びコード変換機能に該当する。

ステップ１３では、共通ブロック要素の参照、例えば、X = VAR のような実行文があったか否かが判定される。そして、共通ブロック要素の値の参照があればステップ１４へと進み（Ｙｅｓ）、共通ブロック要素の値の参照でなければステップ９へと戻る（Ｎｏ）。

【００４８】

ステップ１４では、未定義変数引用の検査のためのライブラリ呼出しが生成される。即ち、ライブラリにより未定義な値の引用かどうかが検査され、プログラム実行時に未定義変数の引用が行なわれると、エラーメッセージが表示されるようになる。その後、ステップ９へと戻る。

【００４９】

以上説明したステップ７～ステップ１４の処理によれば、COMMON文によってプログラム実行時に動的に確保された共通ブロック領域にアクセスできるようにするコードの生成が行なわれる。即ち、スレッド単位でプライベート化する共通ブロックへのアクセスは、ライブラリにより確定されたベース番地に基づいて行なわれるようになる。また、共通ブロック要素の値の参照である場合には、未定義変数引用の検査を行なうことで、プログラムの信頼性を向上することができる。

【００５０】

図４～図８は、ユーザプログラム（以下「プログラム」という）の実行前に行なわれる処理のフローチャートを示す。

先ず、図４は、コンパイラ装置１０のソース解析部１２により生成されたコードによる処理内容を示す。

【００５１】

ステップ２１では、共通ブロックが存在するプログラムの入口で、ステップ８において生成されたライブラリ呼出しが実行される。これにより、制御がライブラリに引き継がれ、後述する処理に従って共通ブロックのベース番地が返却される。

【００５２】

ステップ２２では、ライブラリから返却された共通ブロックのベース番地に基づいて、共通ブロック要素がアクセスされる。

また、図５～図８は、ライブラリにおける処理内容を示す。

【００５３】

ステップ３１では、初期値があることを示す識別子'###-Include initialized values-###'の有無に基づいて、共通ブロック要素に初期値が設定されているか否かが判定される。そして、初期値が設定されていればステップ３２へと進み（Ｙｅｓ）、初期値が設定されていなければステップ３４へと進む（Ｎｏ）。

【００５４】

ステップ３２では、プログラムの実行後に引用された共通ブロック名に基づいて、共通ブロックの引用が初めてであるか否かが判定される。そして、共通ブロックの引用が初めてであればステップ３３へと進み（Ｙｅｓ）、引用が初めてで

なければステップ３４へと進む（Ｎｏ）。

【００５５】

ステップ３３では、ユーザが共通ブロック要素の値を更新してもその初期値が壊されないようにするため、共通ブロックの大きさ分の領域が動的に確保され、そこに共通ブロック要素の値（初期値）が退避される。

【００５６】

以上説明したステップ３１～ステップ３３の処理によれば、共通ブロック要素に初期値が設定されていて、その共通ブロックの引用が初めてであれば、共通ブロック要素の初期値が動的に退避される。

【００５７】

ステップ３４では、プログラムが並列化（マルチスレッド化）されているか否かが判定される。そして、並列化されていればステップ３５へと進み（Ｙｅｓ）、並列化されていなければステップ３８へと進む（Ｎｏ）。

【００５８】

ステップ３５では、実行中のスレッドがＣＰＵ０で実行されているマスタスレッドであるか否かが判定される。そして、かかる条件が成立したならばステップ３８へと進み（Ｙｅｓ）、条件が成立しなければステップ３６へと進む（Ｎｏ）。

【００５９】

ステップ３６では、実行中のスレッドがマスタスレッド以外の並列化されたスレッドであり、かつ、共通ブロックに対するスレッド番号が初めてのものであるか否かが判定される。そして、かかる条件が成立したならばステップ４３へと進み（Ｙｅｓ）、条件が成立しなければステップ３７へと進む（Ｎｏ）。

【００６０】

ステップ３７では、実行中のスレッドがマスタスレッド以外の並列化されたスレッドであって、共通ブロックに対するスレッド番号が設定済みであると判断できるので、スレッド番号に対する共通ブロックのベース番地が返却される。ここで、共通ブロックのベース番地は、ライブラリ内で管理されているスレッド番号に対するベース番地を検索することで特定される。

【００６１】

プログラムが並列化されていない場合の処理を行うステップ３８では、共通ブロックの引用が初めてであるか否かが判定される。そして、共通ブロックの引用が初めてであればステップ３９へと進み（Ｙｅｓ）、引用が初めてでなければステップ４２へと進む（Ｎｏ）。

【００６２】

ステップ３９では、共通ブロック要素に初期値が設定されているか否かが判定される。そして、初期値が設定されていればステップ４２へと進み（Ｙｅｓ）、初期値が設定されていなければステップ４０へと進む（Ｎｏ）。

【００６３】

ステップ４０では、未定義変数引用の検査のためのライブラリ呼出しの有無に基づいて、未定義変数に対する引用の検査の指示があるか否かが判定される。そして、引用の検査の指示があればステップ４１へと進み（Ｙｅｓ）、引用の検査の指示がなければステップ４２へと進む（Ｎｏ）。

【００６４】

ステップ４１では、共通ブロック要素の初期値として、初期値が未設定であることを示す識別子（例えば、「8B8B…8B8B」）が設定される。

ステップ４２では、ユーザ指定のオリジナル共通ブロックのベース番地がそのまま返却される。

【００６５】

実行中のスレッドがマスタスレッドでなく、共通ブロックに対するスレッド番号が初めてのものである場合の処理を行うステップ４３では、スレッド番号に対する共通ブロック領域が動的かつ新規に確保される。

【００６６】

ステップ４４では、共通ブロック名に基づいて、その共通ブロック名に対応する共通ブロック要素の初期値が退避されているか否かが判定される。そして、初期値が退避されていればステップ４５へと進み（Ｙｅｓ）、退避されている初期値が共通ブロック領域に複写される。一方、初期値が退避されていなければステップ４６へと進む（Ｎｏ）。

【００６７】

ステップ４６では、未定義変数に対する引用の検査の指示があるか否かが判定される。そして、引用の検査の指示があればステップ４７へと進み（Ｙｅｓ）、共通ブロック要素の初期値として、初期値が未設定であることを示す識別子（例えば、「8B8B…8B8B」）が設定される。一方、引用の検査の指示がなければステップ４８へと進む（Ｎｏ）。

【００６８】

ステップ４８では、ステップ４３において動的に確保された共通ブロック領域が、ライブラリ内でスレッド番号及び共通ブロック名で管理される。

ステップ４９では、動的に確保された共通ブロック領域のベース番地が返却される。

【００６９】

以上説明したステップ３４～ステップ４９の処理によれば、プログラムが並列化されている場合には、スレッドの実行状態及び共通ブロックの引用状態に応じて、次のような処理が実行される。

【００７０】

(1)実行中のスレッドがマスタスレッドであり、かつ、共通ブロックの引用が初めてである場合

共通ブロック要素に初期値が未設定であり、かつ、未定義変数引用の検査指示があれば、共通ブロック要素の初期値として「8B8B…8B8B」が設定され、ユーザ指定のオリジナル共通ブロックのベース番地が返却される。一方、かかる条件が成立しなければ、ユーザ指定のオリジナル共通ブロックのベース番地がそのまま返却される。

【００７１】

(2)実行中のスレッドがマスタスレッドであり、かつ、共通ブロックの引用が初めてではない場合

ユーザ指定のオリジナル共通ブロックのベース番地がそのまま返却される。

【００７２】

(3)実行中のスレッドがマスタスレッドではなく、かつ、共通ブロックに対す

るスレッド番号が初めてのものである場合

共通ブロック領域が動的に確保され、初期値が退避されていれば、その初期値が共通ブロック領域に複写される。これにより、動的に確保された共通ブロック領域に対して、初期値の設定が動的に行なわれる。一方、初期値が退避されておらず、未定義変数引用の検査指示があれば、共通ブロック要素の初期値として「8B8B…8B8B」が設定される。その後、動的に確保された共通ブロック領域が、スレッド番号及び共通ブロック名で管理され、そのベース番地が返却される。この管理情報を検索することで、共通ブロックが初めて引用されたものか否かを判定したり、共通ブロック名とスレッド番号との関連付けをすることができる。

【００７３】

(4)実行中のスレッドがマスタスレッドではなく、かつ、共通ブロックに対するスレッド番号が設定済みの場合

ライブラリ内で管理されているスレッド番号に対応する共通ブロックのベース番地が返却される。

【００７４】

また、プログラムが並列化されていない場合には、次のような処理が実行される。

共通ブロックの引用が初めてであり、共通ブロック要素に初期値が未設定であり、かつ、未定義変数引用の検査指示があれば、共通ブロック要素の初期値として「8B8B…8B8B」が設定され、ユーザ指定のオリジナル共通ブロックのベース番地が返却される。一方、かかる条件が成立していなければ、ユーザ指定のオリジナル共通ブロックのベース番地が返却される。

【００７５】

従って、ソースプログラムのコンパイル時には、ライブラリにより返却された共通ブロックのベース番地を使用して、共通ブロック要素の引用を行なう方式に変換される。一方、ライブラリにおいては、実行中のスレッドがマスタスレッド以外の並列化されたスレッドである場合、動的に確保された共通ブロック領域のベース番地が返却される。このため、プログラムにおいては、共通ブロック領域が動的に確保されたものか否かにかかわらず、ライブラリから返却されたベース

番地に基づいて共通ブロック要素をアクセスすることができるようになる。

【００７６】

これにより、並列処理においてインターフェース領域を使用した手続呼出しが実現され、アプリケーションプログラムの処理性能を大幅に向上することができるようになる。

【００７７】

なお、共通ブロックのベース番地を返却するライブラリは、必要に応じて動的に呼出される動的ライブラリ（Dynamic Linked Library）でも、リンカによってオブジェクトプログラムと結合される静的ライブラリ（Static Library）のどちらであってもよい。

【００７８】

また、以上説明した実施形態では、ソースプログラムはFORTRANで記述されているものを前提としたが、FORTRAN以外の言語、例えば、C，C++等で記述したものであっても、本発明を適用できることは言うまでもない。

【００７９】

このような機能を実現するプログラムを、例えば、磁気テープ，磁気ディスク，磁気ドラム，ＩＣカード，ＣＤ－ＲＯＭ等の可搬記録媒体に記録しておけば、本発明に係るコンパイラプログラムを市場に流通させることができる。そして、かかる記録媒体を取得した者は、一般的なコンピュータシステムを利用して、本発明に係るコンパイル装置を容易に構築することができる。

【００８０】

【発明の効果】

以上説明したように、請求項１又は請求項２に記載の発明によれば、複数のスレッドが並列処理されるときであっても、各スレッドにおけるインターフェース領域が独立しているため干渉することがなく、従来技術では不可能であった並列処理におけるインターフェース領域を使用した手続呼出しを実現することができる。

【００８１】

請求項３又は請求項７に記載の発明によれば、ソースプログラム中にインター

フェース領域の先頭番地を確定するための処理を記述する必要はなく、プログラム構造を簡単にすることができる。

【００８２】

請求項４又は請求項８に記載の発明によれば、ユーザの考えに即した、柔軟なプログラム構造を提供することができる。

請求項５又は請求項６に記載の発明によれば、請求項１又は請求項２に記載の発明の効果に加え、本発明に係るコンパイラプログラムを市場に流通させることができる。そして、かかる記録媒体を取得した者は、一般的なコンピュータシステムを使用して、本発明に係るコンパイラ装置を容易に構築することができる。

【図面の簡単な説明】

【図１】

本発明に係るコンパイラ装置の構成図である

【図２】

ソース解析部による並列処理を実現するための処理内容を示すフローチャートである。

【図３】

ソース解析部による並列処理を実現するための処理内容を示すフローチャートである。

【図４】

ソース解析部により生成されたコードを実行するときの処理内容を示すフローチャートである。

【図５】

ライブラリにおける処理内容を示すフローチャートである。

【図６】

ライブラリにおける処理内容を示すフローチャートである。

【図７】

ライブラリにおける処理内容を示すフローチャートである。

【図８】

タスク並列処理における概要図であり、（Ａ）は逐次処理のソースプログラ

ム、（Ｂ）は並列処理のソースプログラム、（Ｃ）は各ＣＰＵにおいて実行される手続を夫々示す。

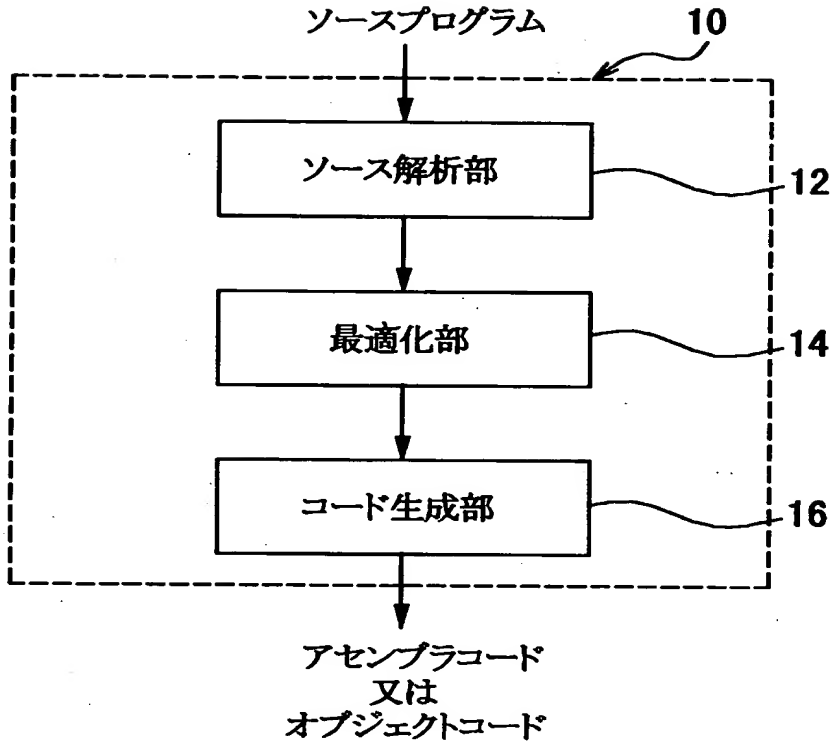
【符号の説明】

１０…コンパイラ装置

１２…ソース解析部

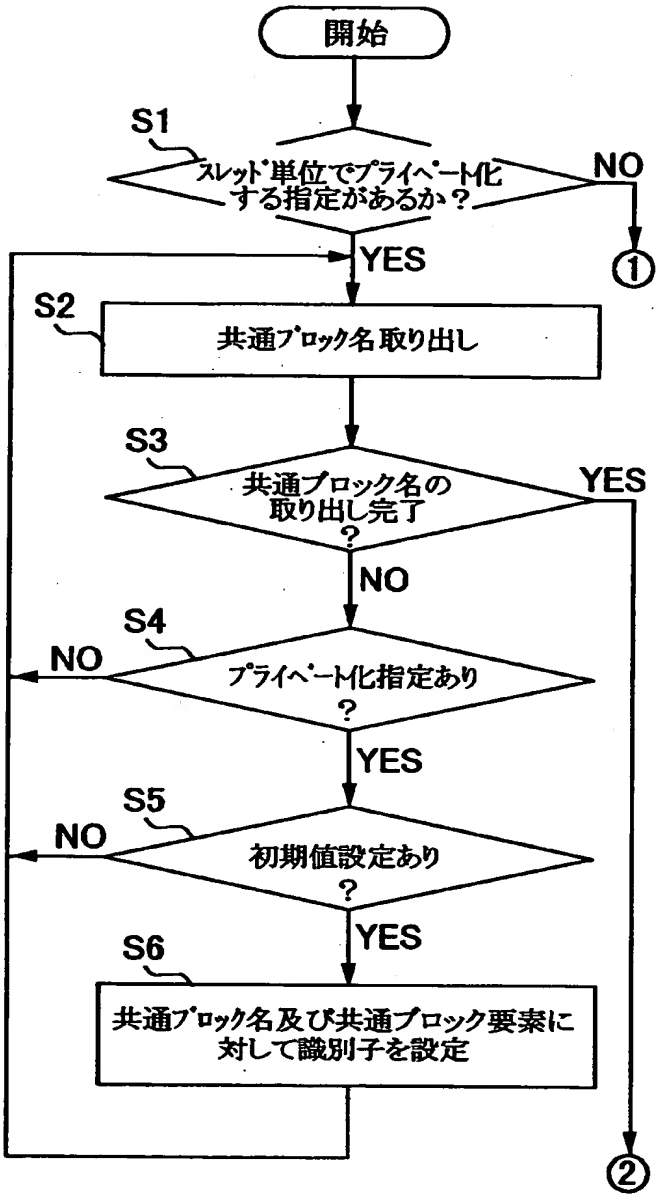
【書類名】　　図面

【図１】

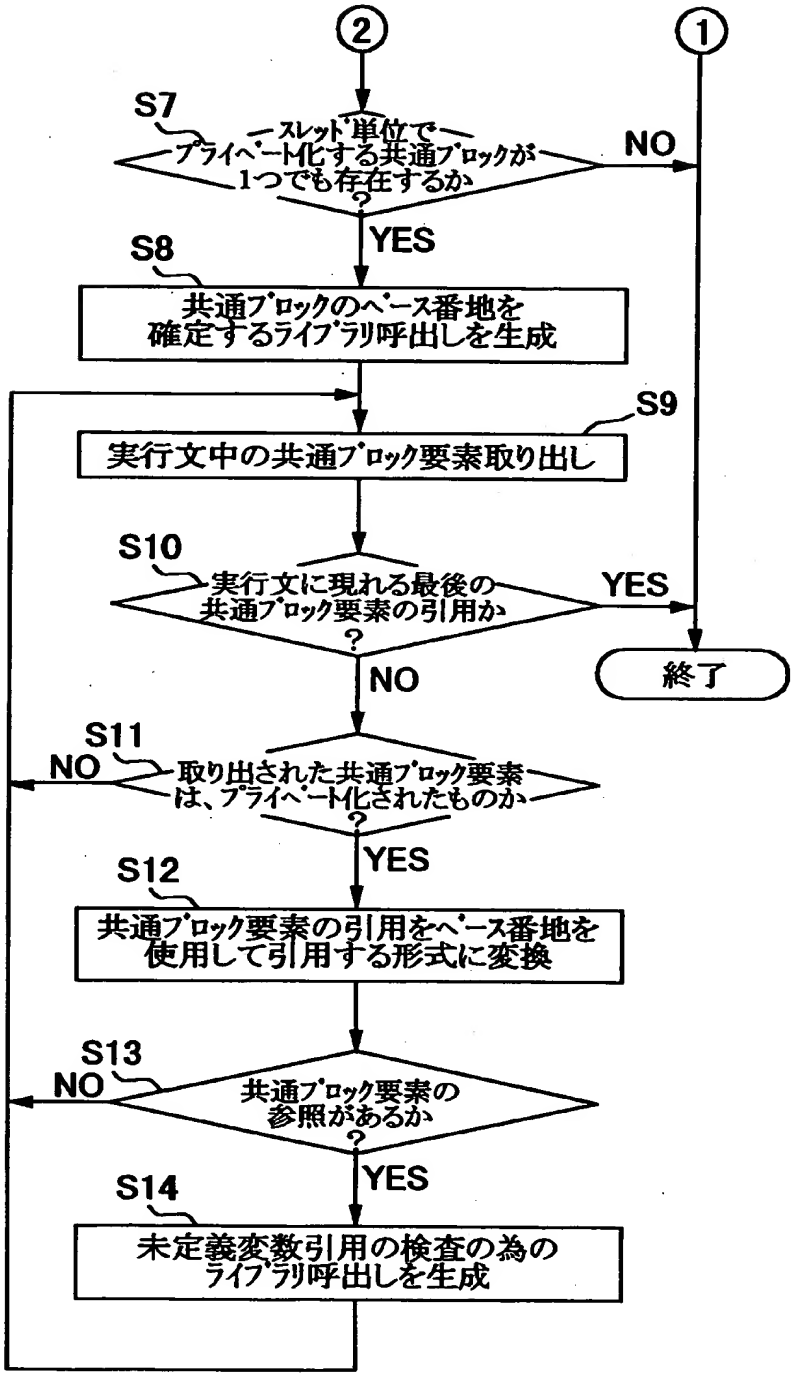


10 コンパイラ装置
12 ソース解析部

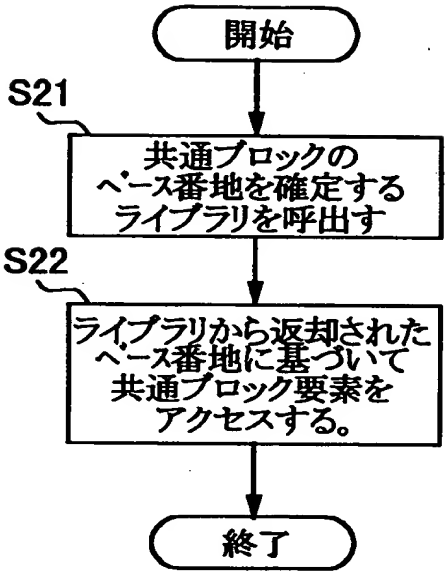
【図２】


開始
S1
スレッド単位でプライベート化する指定があるか？
NO
①
YES
S2
共通ブロック名取り出し
S3
共通ブロック名の取り出し完了？
YES
NO
S4
NO
プライベート化指定あり？
YES
S5
NO
初期値設定あり？
YES
S6
共通ブロック名及び共通ブロック要素に対して識別子を設定
②

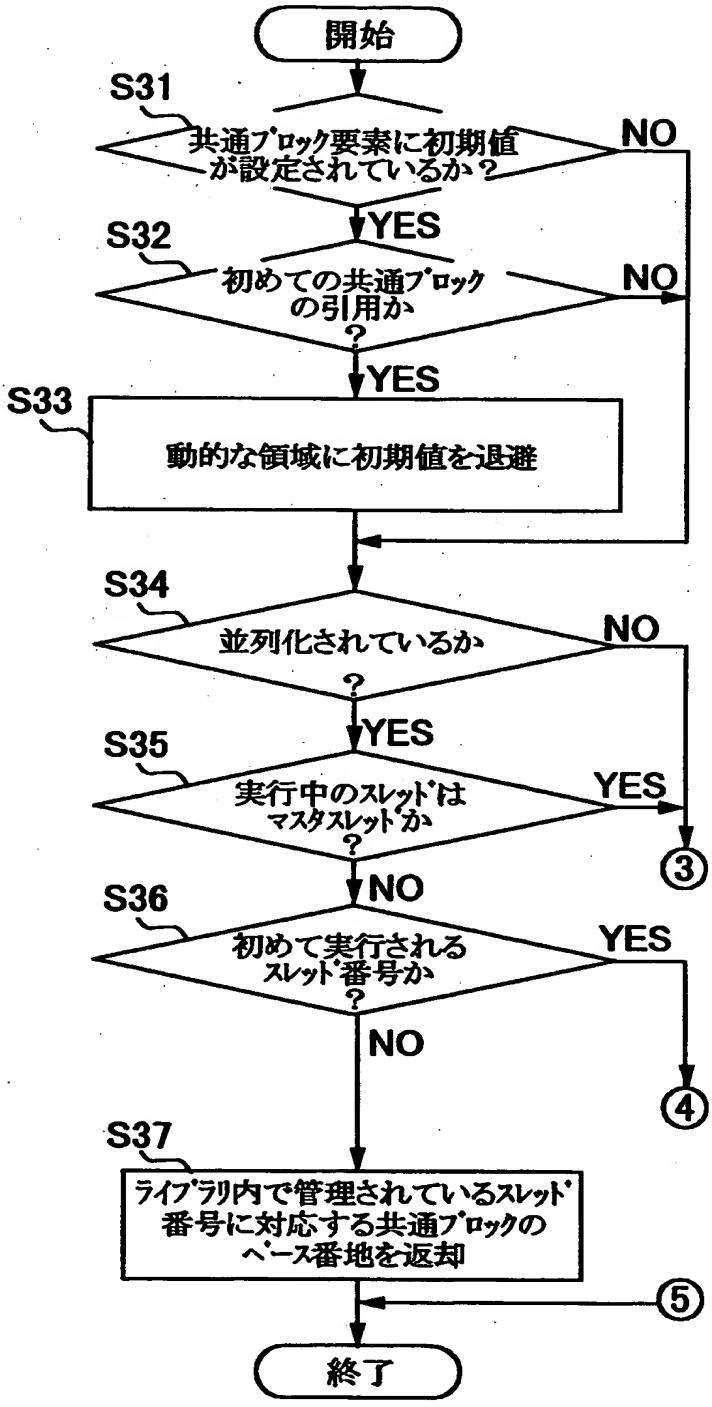
【図３】


2
1
S7
スレッド単位で
プライベート化する共通ブロックが
1つでも存在するか
?
NO
YES
S8
共通ブロックのベース番地を
確定するライブラリ呼出しを生成
S9
実行文中の共通ブロック要素取り出し
S10
実行文に現れる最後の
共通ブロック要素の引用か
?
YES
終了
NO
S11
NO
取り出された共通ブロック要素
は、プライベート化されたものか
?
YES
S12
共通ブロック要素の引用をベース番地を
使用して引用する形式に変換
S13
NO
共通ブロック要素の
参照があるか
?
YES
S14
未定義変数引用の検査の為の
ライブラリ呼出しを生成

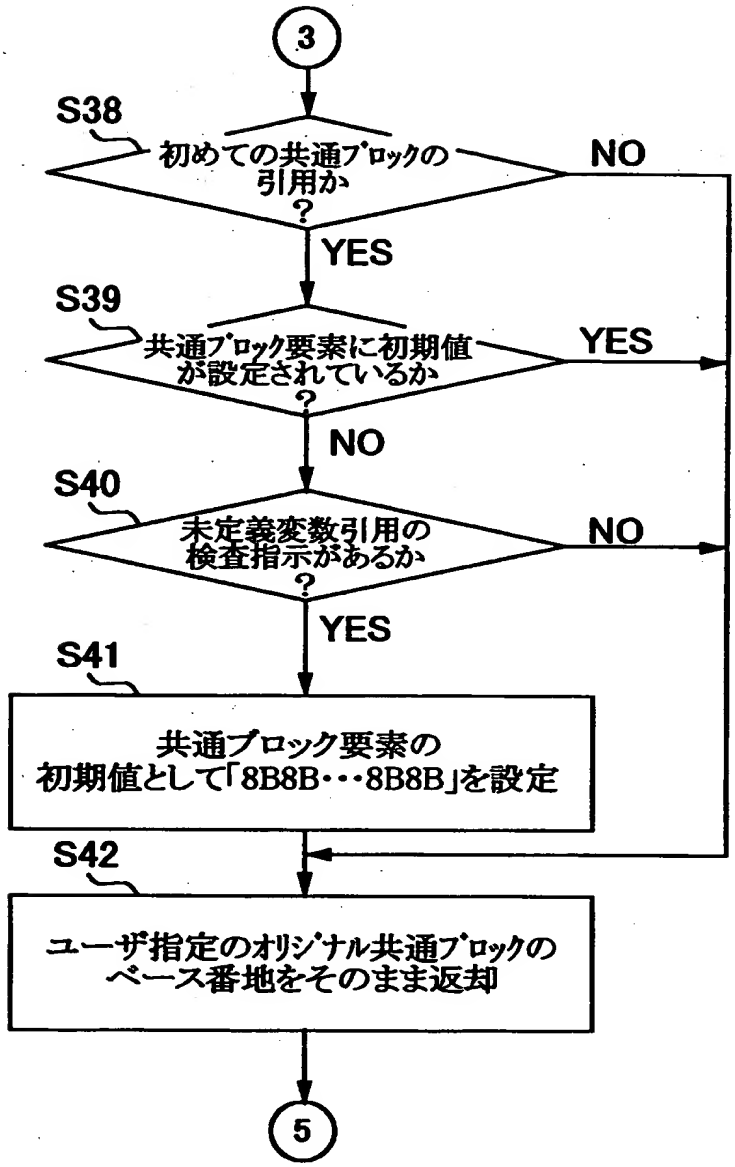
【図４】


開始
S21
共通ブロックの
ベース番地を確定する
ライブラリを呼出す
S22
ライブラリから返却された
ベース番地に基づいて
共通ブロック要素を
アクセスする。
終了

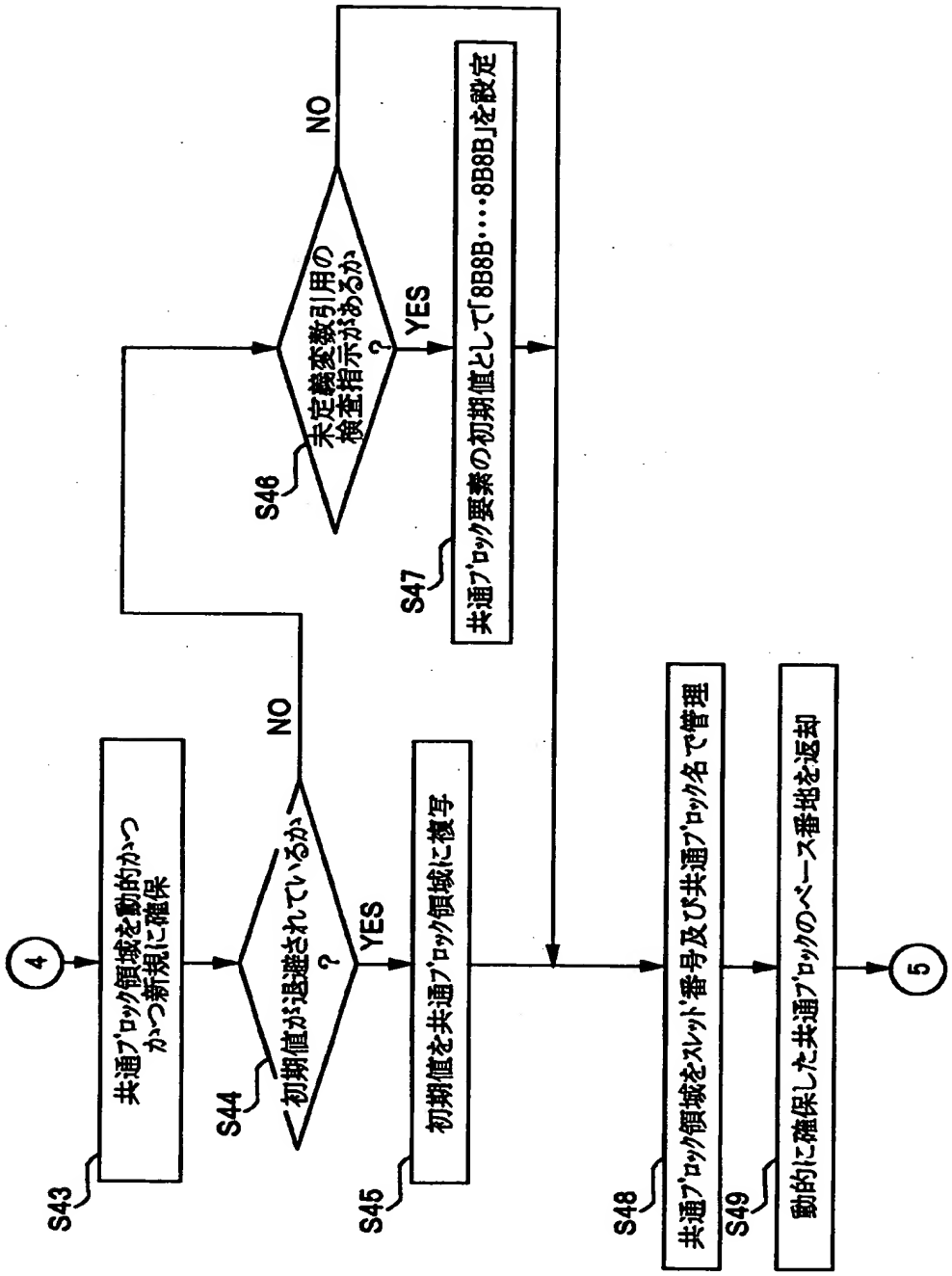
【図５】


開始
S31
共通ﾌﾞﾛｯｸ要素に初期値が設定されているか？
NO
YES
S32
初めての共通ﾌﾞﾛｯｸの引用か？
NO
YES
S33
動的な領域に初期値を退避
S34
並列化されているか？
NO
YES
S35
実行中のｽﾚｯﾄﾞはﾏｽﾀｽﾚｯﾄﾞか？
YES
③
NO
S36
初めて実行されるｽﾚｯﾄﾞ番号か？
YES
④
NO
S37
ﾗｲﾌﾞﾗﾘ内で管理されているｽﾚｯﾄﾞ番号に対応する共通ﾌﾞﾛｯｸのﾍﾞｰｽ番地を返却
⑤
終了

【図６】


3
S38
初めての共通ブロックの引用か？
NO
YES
S39
共通ブロック要素に初期値が設定されているか？
YES
NO
S40
未定義変数引用の検査指示があるか？
NO
YES
S41
共通ブロック要素の初期値として「8B8B･･･8B8B」を設定
S42
ユーザ指定のオリジナル共通ブロックのベース番地をそのまま返却
5

【図７】


4
S43
共通ブロック領域を動的かつ
かつ新規に確保
S44
初期値が退避されているか
？
NO
YES
S45
初期値を共通ブロック領域に複写
S46
未定義変数引用の
検査指示があるか
？
NO
YES
S47
共通ブロック要素の初期値として「8B8B・・・8B8B」を設定
S48
共通ブロック領域をスレッド番号及び共通ブロック名で管理
S49
動的に確保した共通ブロックのベース番地を返却
5

【図８】

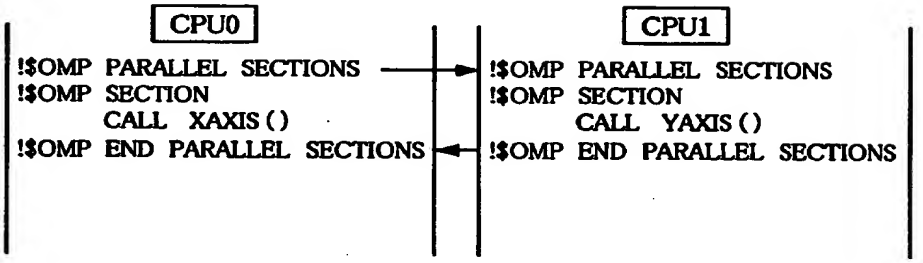
(A)

```
      ⋮
CALL  XAXIS ()
CALL  YAXIS ()
      ⋮
```

(B)

```
      ⋮
!$OMP PARALLEL SECTIONS
!$OMP SECTION
      CALL  XAXIS ()
!$OMP SECTION
      CALL  YAXIS ()
!$OMP END PARALLEL SECTIONS
      ⋮
```

(C)



【書類名】　要約書

【要約】

【課題】　並列処理においてインターフェース領域を使用した手続呼出しを実現する。

【解決手段】　プログラム実行時に、スレッド単位毎に動的に確保された共通ブロック（インターフェース領域）のベース番地（先頭番地）を確定するライブラリ呼出しを生成し（Ｓ８）、ソースプログラム中の共通ブロックの変数等に対する引用を、ライブラリ呼出しを実行することで確定したベース番地を使用して引用する形式に変換する（Ｓ１２）。即ち、動的に確保された共通ブロック領域は、実際に確保されるまでその番地が確定しないので、ライブラリ呼出しによりベース番地を確定し、確定したベース番地に基づいて共通ブロックをアクセスするようにする。従って、複数のスレッドが並列処理されるときであっても、各スレッドにおけるインターフェース領域が独立しているため干渉することがなく、並列処理におけるインターフェース領域を使用した手続呼出しが実現される。

【選択図】　図３

出　願　人　履　歴　情　報

識別番号　　〔０００００５２２３〕

１．変更年月日　　１９９６年　３月２６日
［変更理由］　住所変更
住　所　　神奈川県川崎市中原区上小田中４丁目１番１号
氏　名　　富士通株式会社